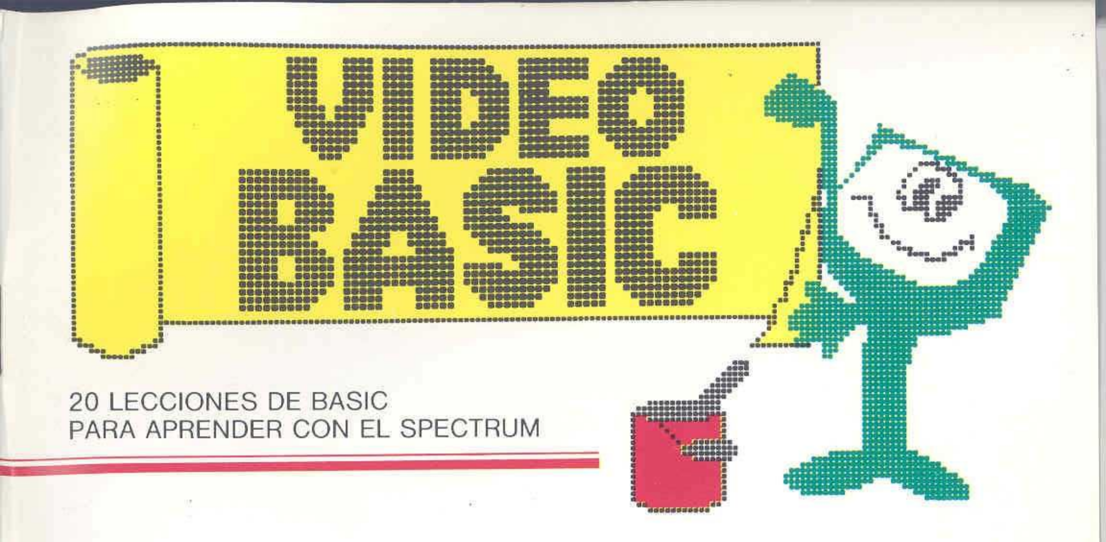


20 LECCIONES DE BASIC
PARA APRENDER CON EL SPECTRUM



- Los ordenadores del futuro
- La inteligencia artificial
- El BASIC y la memorización de los programas
- Las interrupciones
- Usar la ROM
- Videoejercicios
- Videojuego N.º 20

20 Spectrum

16K/48K/PLUS

VIDEO BASIC
Una publicación de
INGELEK JACKSON
Director editor por INGELEK:
Antonio M. Ferrer
Director editor por JACKSON HISPANIA:
Lorenzo Bertagnolio
Director de producción:
Vicente Robles
Autor: Softidea
Redacción software italiano:
Francesço Franceschini,
Stefano Cremonesi
Redacción software castellano:
Fernando López, Antonio Carvajal,
Alberto Caffarato, Pilar Manzanera
Diseño gráfico:
Studio Nuovaidea
Ilustraciones:
Cinzia Ferrari, Silvano Scolari,
Equipo Galata
Ediciones INGELEK, S. A.
Dirección, redacción y administración,
números atrasados y suscripciones:
Avda. Alfonso XIII, 141
28016 Madrid. Tel. 2505820
Fotocomposición: Espacio y Punto, S. A.
Imprime: Gráficas Reunidas, S. A.

ISBN del tomo 4: 84-85831-20-9
ISBN del fascículo: 84-85831-11-X
ISBN de la obra completa: 84-85831-10-1
Depósito Legal: M-15076-1985
Plan general de la obra:
20 fascículos y 20 casetes, de aparición quincenal, coleccionables en 5 estuches.
Distribución en España:
COEDIS, S. A.
Valencia, 245. 08007 Barcelona.
INGELEK JACKSON garantiza la publicación de todos los fascículos y casetes que componen esta obra y el suministro de cualquier número atrasado o estuche mientras dure la publicación y hasta un año después de terminada.
El editor se reserva el derecho de modificar el precio de venta del fascículo, en el transcurso de la obra, si las circunstancias del mercado así lo exigen.
Diciembre, 1985
Impreso en España.



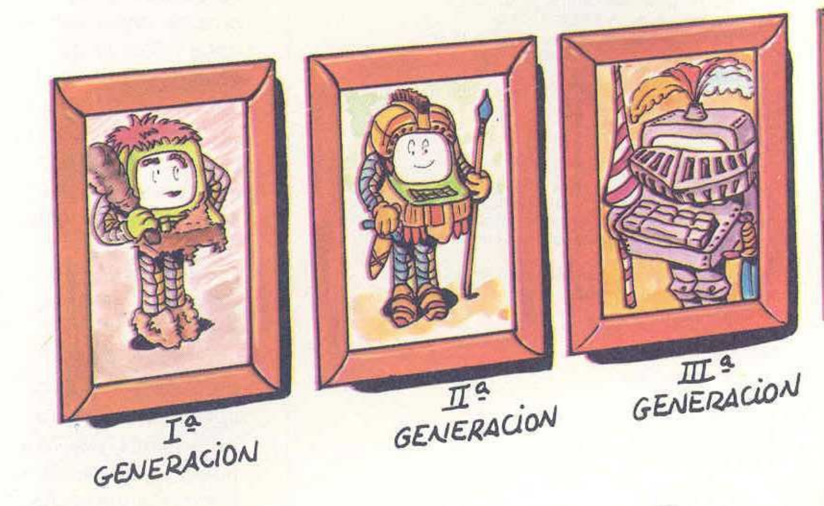
SUMARIO

Introducción

Tu Spectrum posee cualidades y precios impensables hace tan sólo unos años; lo que los laboratorios de investigación están preparando ahora, sin embargo, es algo muy distinto: ordenadores ultrarrápidos, capaces de elaborar más informaciones en paralelo, pero, sobre todo, ordenadores "inteligentes".
La ciencia-ficción, en resumen, ya no pertenece al futuro.
En cualqueir caso, para mantenernos con los pies en el presente debemos perfeccionar cuanto sabemos y aprender cosas nuevas.
Aquí tienes, por tanto, cómo ahorrar tiempo y memoria, los punteros, las interrupciones y todavía un poco más de código máquina.
Paciencia. Si el ordenador todavía no razona y habla igual que un hombre es necesario saber razonar y hablar como el ordenador.

El ordenador futuro

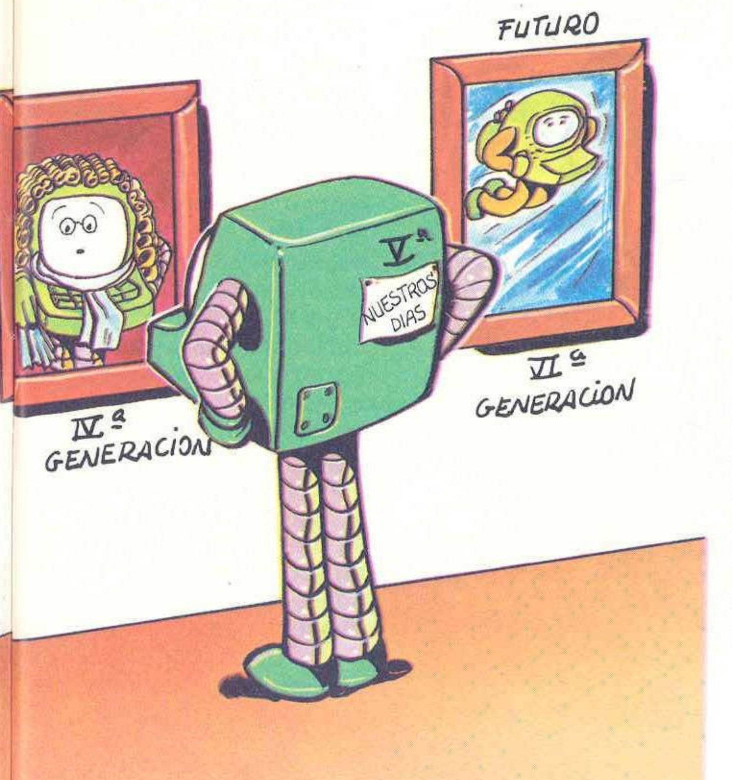
Aunque muchas personas crean todavía lo contrario, los ordenadores no trabajan en absoluto por magia: ya sabemos perfectamente que todos los resultados obtenibles mediante un ordenador electrónico no son otra cosa que el producto de un complejo y rapidísimo conjunto de operaciones elementales, desarrollados en el interior de los circuitos y de las memorias de la máquina. Es decir, que lo que a algunos puede todavía parecer fantástico, es en realidad un concreto y lógico desarrollo de un sector tecnológico que se apoya en sólidas y rigurosas bases teóricas y científicas. Hace tan sólo algunos decenios nadie habría podido pronosticar el nacimiento y, sobre todo, el desarrollo de una tecnología tan revolucionaria como la electrónica: no existía ningún precedente que permitiese entrever un futuro tan cargado de desarrollos. Ahora que



vivimos de pleno en la llamada «era electrónica» poseemos, sin embargo, suficientes «horizontes» para poder imaginar cuáles serán los progresos más probables en el sector de los ordenadores. Los caminos que en estos momentos recorren por los estudiosos de todo el mundo merecen de cualquier manera ser descritos y analizados.

Los ordenadores del pasado

Pero antes de pensar en el futuro demos una ojeada al pasado: el dicho «preparaos para el futuro mirando en el pasado» es más apropiado que nunca para quien, como nosotros, desea fundamentar sus propias hipótesis de una forma concreta.



Dejando de lado los estudios y las teorías lógicas que han permitido el nacimiento y la evolución del cálculo automático llegamos al comienzo de los años 50, cuando todavía imperaban las válvulas y los diodos en el campo electrónico. Hacía poco que la guerra había terminado y los ordenadores estaban apenas en los comienzos: en aquel período empezaron a aparecer las primeras máquinas electrónicas capaces de ejecutar automáticamente cálculos y operaciones matemáticas.
Estos ordenadores parecían dinosaurios comparados con los actuales, con dimensiones poco más o menos enormes (el equivalente a un ordenador personal normal ocupaba un laboratorio entero), compuestos por kilómetros de cables y millares de válvulas. Además consumían notables cantidades de energía eléctrica. Ahora son llamados «ordenadores de la primera generación».

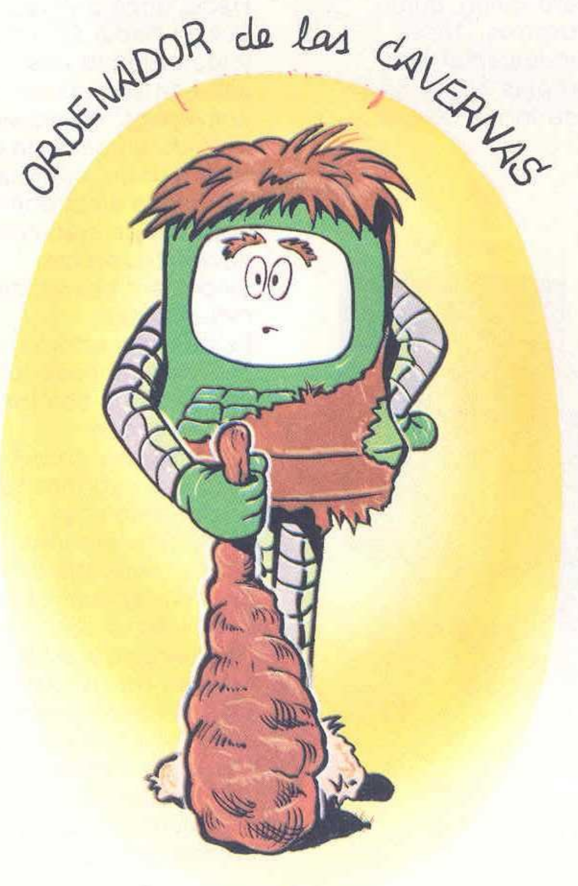
HARDWARE

Las máquinas de este tipo estaban fundamentalmente a disposición de grandes centros de investigación militares o universitarios y de grandes industrias: el hardware era demasiado aparatoso y susceptible de averías como para garantizar inmediatos desarrollos o aplicaciones comerciales. Respecto al software las cosas no iban mucho mejor: el código máquina y los lenguajes simbólicos elementales del tipo ensamblador imponían grandes limitaciones de uso.

Pero, poco a poco, los ordenadores comenzaron a difundirse: era necesario, sin embargo, hacerlos más fiables y menos engorrosos. El hecho decisivo fue el descubrimiento del transistor. Un transistor desempeña en un ordenador la misma función que una válvula: el transistor consume, sin embargo, menos energía que las válvulas, es de dimensiones mucho menores y, sobre todo, su vida media es mucho más larga. Además y este hecho es también esencial, resulta más barato.

A pesar de ser sustancialmente parecidos a los ordenadores anteriores en lo referente a su lógica, estos nuevos sistemas (llamados de la segunda generación) se diferencian de ellos en varios aspectos, sobre todo en sus dimensiones.

Comenzaba una verdadera revolución y, con ella, el período del



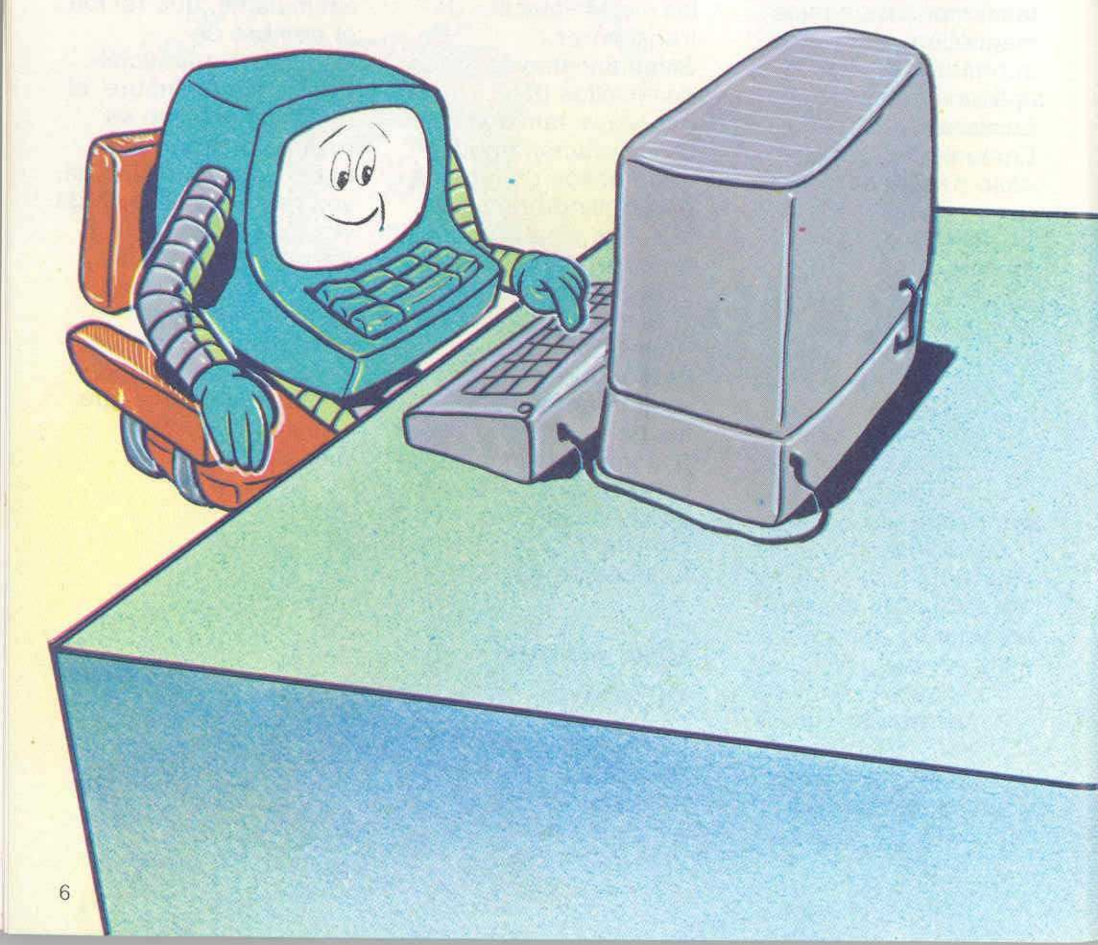
auténtico «despegue» del ordenador. Muchas empresas comprendieron la practicidad y la utilidad de un ordenador electrónico, y lo instalaron pidiendo al mismo tiempo una mayor potencia y una velocidad de ejecución aún mayor. Nuevas posibilidades de elaboración aparecieron con la introducción de la memoria de núcleo magnético, ampliando aún más las aplicaciones y los posibles desarrollos. En cualquier caso, el paso hacia adelante más decisivo consistió en la creación de circuitos integrados o chips: esta nueva tecnología generó los ordenadores de la «tercera generación». Los circuitos integrados introdujeron nuevas mejoras, miniaturizando y «refinando» los componentes de la segunda generación. Además de esto, eran aún más baratos que las viejas «placas» de transistores. Simultáneamente, a los desarrollos del hardware, también la programación había dado pasos de gigante, proponiendo nuevas técnicas y aplicaciones gracias a lenguajes cada vez más potentes y, al mismo tiempo, más «humanos». Los ordenadores de la «cuarta generación» son los de nuestros días: pequeños, eficientes, fiables, pero —a pesar de las apariencias— todavía muy mejorables. Veamos en qué forma.

Una ojeada al futuro

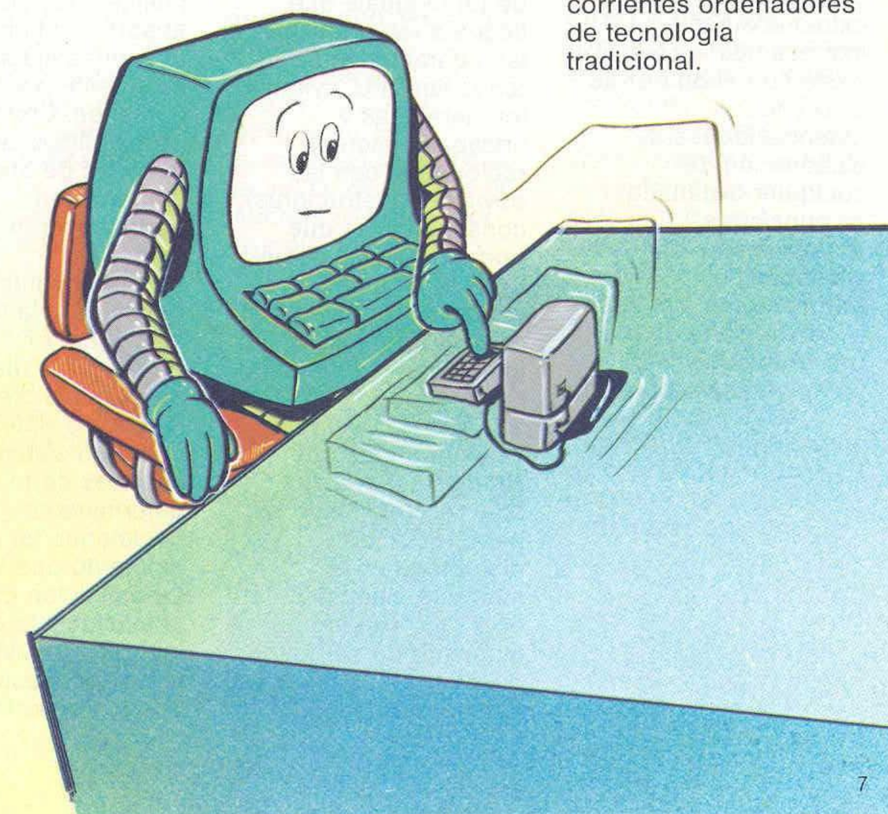
La investigación científica se halla ya en un estado avanzado: se están recorriendo muchos caminos, tanto en «hardware» como en «software». Las principales áreas de estudio desde el punto de vista constructivo se refieren, sobre todo, a las superconductividad y la elaboración paralela, mientras que la informática propiamente dicha vuelca todas sus esperanzas en el sector de investigación, extremadamente estimulante, que recibe el nombre de «inteligencia artificial». Como de costumbre, el principal objetivo es realizar ordenadores cada vez mejores, cada vez más versátiles, cada vez más útiles; que trabajen más deprisa, que memoricen más información, que necesiten menos potencia, que ocupen menos espacio, y que cuesten cada vez menos.

Los ordenadores con superconductores

En realidad, los ordenadores con superconductores existen ya desde hace algún tiempo: sin embargo, las posibilidades de desarrollo que parecen capaces de ofrecer les colocan en un sector que pertenece más al mañana que al hoy.
Los ordenadores que se construyen actualmente han alcanzado velocidades de elaboración tan elevadas que pueden ser comparadas con aquéllas con las que los electrones se mueven en los circuitos electrónicos. Es evidente que si los desplazamientos de los electrones en el interior del ordenador (recordemos que el flujo de los electrones en el interior de los circuitos constituye la corriente eléctrica) son más lentos que las posibilidades de cálculo de la unidad central, el tiempo de ejecución está abocado a disminuir.
En otras palabras, las



informaciones emplean un cierto tiempo (aunque sea pequeñísimo) para ir de un punto al otro de los circuitos electrónicos: si este tiempo es superior al de la velocidad de cálculo de la CPU, ésta será «ralentizada» a la fuerza, con consecuencias obvias para la velocidad de trabajo de todo el sistema.
Este problema, a primera vista aparentemente irresoluble (no existe ninguna posibilidad de «acelerar» el movimiento de los electrones en los conductores, siendo este último una característica específica de los materiales), se ha resuelto recurriendo a determinadas aleaciones de metales conductores, que gozan de la especial propiedad de oponer —a temperaturas extremadamente bajas (más de 100 grados bajo cero)— una resistencia al movimiento mucho más baja de la que ofrecen a temperaturas ambientales normales.
El problema se resuelve así brillantemente: las velocidades de elaboración de los ordenadores con superconductores (superconductividad es el nombre del proceso físico que acabamos de ver) alcanzan valores absolutamente impensables en los corrientes ordenadores de tecnología tradicional.



Los ordenadores paralelos

La idea que sirve de base a la técnica de los ordenadores paralelos es de una sencillez casi apabullante: en vez de usar una sola unidad central, en los ordenadores paralelos se usan más unidades centrales, que trabajan de manera simultánea, o, como se dice más frecuentemente, en paralelo. Las posibilidades que se proponen son extremadamente interesantes: en teoría basta con añadir otras CPU y las potencialidades de elaboración de cualquier ordenador se convierten prácticamente en ilimitadas.
Naturalmente, como en la mayor parte de los proyectos que aparentemente parecen «sencillos», los espacios de acción que separan el dicho del hecho resultan muy delicados de resolver.
Los principales problemas de los ordenadores paralelos no residen únicamente en las propias disposiciones circuitales (cualquier cosa menos fáciles de resolver) sino también (y sobre todo) en la modalidad de programación que estas máquinas requieren.
Es necesario disponer de un lenguaje que llegue a «sincronizar» las operaciones de todas las CPU, evitando interferencias y encabalgamientos recíprocos, con las obvias (y destructoras) consecuencias que podrían derivar de esto.
En este momento, no existe ningún lenguaje de este tipo: los intentos experimentados hasta ahora dejan entrever en cualquier caso algo más que simples esperanzas.
Muy recientemente se han producido y distribuido en el mercado máquinas de este tipo, aunque por el momento no se puede disfrutar plenamente de sus posibilidades.

Inteligencia artificial

La inteligencia artificial es, sin duda alguna, el sector en el que el desarrollo de la ciencia de los ordenadores tendrá mayor influencia sobre nuestro modo de vivir en los próximos decenios.
Durante generaciones, los escritores de ciencia-ficción han pronosticado la evolución de máquinas más o menos inteligentes, capaces de absorber muchas de las funciones ejecutables solamente por los seres humanos. Con toda probabilidad las historias de androides y humanoides permanecerán, sin embargo, exclusivamente en el dominio de la literatura fantástica y futurológica.
En nuestros días se piensa más bien en máquinas «inteligentes» como en sistemas capaces de tomar determinadas decisiones en el momento oportuno.
La definición exacta de la inteligencia de una máquina todavía es un tema por resolver, dado que los expertos siguen

enzarzados en constantes debates sobre el tema; de cualquier forma se puede aceptar como definición estándar la que propuso en los «lejanos» 40 un auténtico pionero de la informática: Alan Turing.
Más que enumerar una serie de criterios a satisfacer para poder clasificar un ordenador como inteligente, se limitó a dar una opinión mucho más práctica sobre el problema.
Turing afirmó que si una persona no era capaz de distinguir si determinadas respuestas le llegaban de una máquina o de otra persona, había que afirmar entonces que la máquina que eventualmente hubiera elaborado tales respuestas tandría que clasificarse como inteligente. Esta prueba constituye la base del famoso «test de Turing», en el cual un operador humano tiene que sostener a través de un teclado y un terminal una determinada conversación, intentando obligar al interlocutor a desvelar su propia identidad de hombre o máquina.
Los centros de investigación de todo el mundo están llevando a cabo estudios y pruebas sobre la inteligencia artificial, y, también en este caso, parece que llegar a resultados efectivos es sólo una cuestión de años. Japón ya se ha adelantado a todos los demás, anunciando que su ordenador de la «quinta generación» verá la luz como máximo en 1995.
La inteligencia artificial ya se ha implantado en muchos sectores, por ejemplo, en el de los «sistemas expertos»: con este nombre definimos aquellos ordenadores especializados que son capaces de ejecutar una determinada tarea con iguales (y a veces mejores) resultados que el de los expertos humanos.
Un ejemplo típico del uso de esta tecnología lo vemos todos los días al ver las previsiones del tiempo, elaboradas cotidianamente mediante la ayuda precisamente de los mencionados sistemas expertos. También en el campo del ordenador-doctor ya no es algo aventurado.
La mayor parte de los estudios sobre inteligencia artificial se realizan empleando lenguajes especiales, creados especialmente para ello (generalmente LISP y PROLOG).
Puesto que estos lenguajes requieren potencias de cálculo muy superiores a las ofrecidas por los pequeños ordenadores, la inteligencia artificial entrará en nuestras casas dentro de pocos años solamente gracias a la telemática, que permitirá conectar el ordenador doméstico a un gran sistema inteligente.
Sin embargo, esto no significa que el campo de la inteligencia artificial no pueda afrontarse, aunque sea a nivel aficionado, desde un pequeño ordenador personal.

Ahorrar tiempo y memoria

En la programación, como en otros muchos sectores de la actividad humana, se ha ido formando una escala de valores que clasifica las diferentes técnicas como «buenas», «no muy buenas», o «pésimas».
Cuando empezaste a usar tu Spectrum, escribir un programa que funcionara ya era de por sí un resultado válido. Ahora que eres mucho más dueño de la situación, y dispones de conocimientos y capacidades que en un principio no tenías, ha llegado el momento de tocar un tema que, a primera vista, te puede parecer poco relevante, pero que en realidad tiene una enorme importancia para mejorar tu técnica de programación.
Queremos pues ver lo que habrá que hacer para mejorar y aumentar la velocidad de ejecución de tus programas BASIC, sin que esto influya en su calidad, su legibilidad y no aumente la cantidad de memoria ocupada.
No siempre la velocidad de ejecución y el ahorro de memoria resultan compatibles: existen, sin embargo, un buen número de «truquitos», que pueden ser útiles. Veamos pues algunos de ellos:

- **Instrucciones múltiples.**

El situar más de una instrucción en una misma línea ayuda a reducir la longitud del programa, ahorrando números de línea y en consecuencia ocupación de memoria.
La desventaja y la limitación principal de esta técnica reside en la más escasa legibilidad y en la difícil modificabilidad de las distintas líneas. Por lo tanto, es mejor no abusar de esta práctica.

- **Variables.**

Las variables tendrían que tener nombres lo más cortos posibles.
Esto ayuda a ahorrar memoria y acelera el trabajo del intérprete BASIC.
También, las constantes (tanto múltiples como alfanuméricas), si son usadas con frecuencia, conviene asignarlas a variables. Por ejemplo, en lugar de escribir:

```
10 POKE 15143,12:POKE 15143,70:POKE 15143,20
```

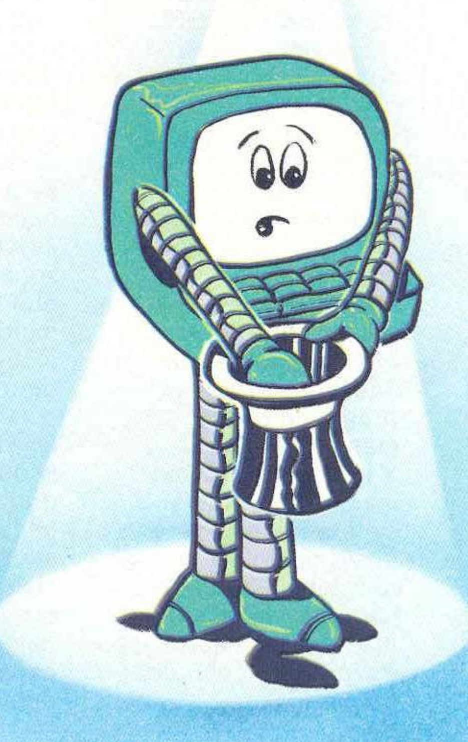
convendría hacer:

```
10 LET A=15143:POKE A,12:POKE A,70:POKE A,20
```

Así, el intérprete BASIC tendrá que convertir una sola vez el número 15143 a un formato comprensible por la CPU, reduciendo además el espacio ocupado en memoria.

- **REM.**

Es necesario limitar al máximo el uso de los comentarios en los programas. Esta indicación parece una contradicción con respecto a todo lo anteriormente comentado: es decir, que la documentación de los programas debería ser lo más abundante posible. Los programadores expertos resuelven este problema conservando dos copias del mismo programa: la primera, comentada en todos sus aspectos, sirve para comprender el funcionamiento del programa y para poder aportar toda clase de posibles modificaciones; la segunda, que tendrá que funcionar en el ordenador, será privada de todo comentario, para evitar cualquier gasto «inútil» de memoria.



• **GOTO.**
Cada vez que el intérprete tiene que ejecutar un salto, la secuencia de las instrucciones sufre un cambio imprevisto, desplazándose a otro punto de programa.
La entidad de este cambio se define especificando en la instrucción GOTO el número de línea al que saltar. Pero el intérprete BASIC no dispone de especiales técnicas de búsqueda para localizar este número de línea y ejecuta entonces una lenta búsqueda secuencial desde el principio del programa.
Por lo tanto, el tiempo necesario para ejecutar una instrucción de salto depende de la longitud del texto y de la distancia desde la línea indicada en el GOTO hasta el principio del texto.
Cuando se desee acelerar al máximo la velocidad de ejecución, será necesario considerar cuidadosamente este hecho, intentando limitar todo lo posible la existencia de GOTO (lo que además aumenta la legibilidad del programa) o, si realmente no se puede prescindir de ellos, acercando al máximo las instrucciones a las que hay que saltar, al principio del programa.

• **GOSUB.**
El empleo de subrutinas permite un notable ahorro de memoria, dado que evita la escritura repetida de grupos de instrucciones. Sin embargo, para las instrucciones GOSUB sirven las mismas consideraciones hechas para GOTO; lo mejor, y no se trata de algo difícil, es colocar todas las subrutinas al principio del programa en lugar de al final, tal y como estamos acostumbrados dándole preferencia, es decir, escribiéndolas antes, a aquéllas que el programa emplee con más frecuencia.
Y la recomendación más importante es siempre la misma: es decir, cualquier programa se puede hacer notablemente más rápido y corto si es estructurado con una buena lógica.

Cómo trabaja el BASIC: los punteros

Un puntero es una porción de la memoria del ordenador (normalmente muy pequeña: una o dos localizaciones) cuyo contenido está constituido por direcciones útiles para el funcionamiento del sistema. Nos explicaremos mejor con un ejemplo.
Cuando enciendes tu Spectrum, el intérprete BASIC tiene que estar inmediatamente en condiciones de aceptar las líneas de programas que desees teclear.
Para poder hacer esto será evidentemente necesario que el intérprete conozca la zona de memoria en la que tendrá que memorizar las distintas instrucciones; la dirección de esta zona estará contenida pues en el puntero adecuado, leído por el intérprete durante la rutina de encendido (o inicialización) del sistema. En cada ordenador existen numerosos punteros, cada uno de ellos con una función específica.

Su principal utilidad es que gracias a ellos es posible referirse con un mínimo trabajo a determinadas áreas de la memoria, lo que permite que se puedan introducir con toda facilidad posibles modificaciones o puestas al día.
En breve descubriremos que también la memorización de las líneas de programa recurre ampliamente a esta técnica.



La memorización de los programas

Cuando tecleas en BASIC una línea de programa, ésta se memorizará de esta forma:

- 2 bytes para el número de línea, empleados de tal modo que el byte más significativo preceda al byte menos significativo.
- 2 bytes como contadores de la longitud de la línea, desde el primer carácter del texto hasta el final de la línea (comprendido el carácter ENTER); en este caso los dos bytes se emplean en el modo byte bajo-byte alto.
- El texto de la palabras BASIC escritas en código ASCII, donde:

– las palabras y los símbolos reservados ocupan un solo byte. Para ahorrar memoria, las palabras reservadas son convertidas en determinados códigos numéricos, llamados TOKEN. Así, por ejemplo, la palabra GOTO no se memoriza con cuatro letras separadas («G», «O», «T», «O»), es decir, en 4 bytes, sino con el único código 236;
– los parámetros y los signos de puntuación se presentan, carácter por carácter, en código ASCII;
– los parámetros numéricos se representan de dos formas distintas; como cadenas de números y como números en formato exponencial. La cadena se usa durante el listado, es decir, cuando desees leer lo que has escrito, mientras que el formato exponencial se usa durante la ejecución del programa (esta es una forma extremadamente ingeniosa para ahorrar al intérprete el trabajo de convertir constantemente los distintos números). Las dos formas se usan separadas por un código numérico (14), de forma que el intérprete siempre sepa cuál debe elegir.

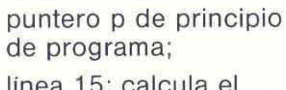
- El código 13 para ENTER.

Para indicar el final del programa no se usa ningún código especial. El intérprete simplemente indica si el byte siguiente al código ENTER tiene los primeros dos bits de la izquierda en 01,10 o 11: si es así seguramente no se trata de una nueva línea de programa.
Sigue un programa que ilustra todo lo indicado. Este se «lee» a sí mismo en la memoria del Spectrum.


SISTEMA
OPERATIVO
BZZZZ
BZZZZ
TAC TAC TAC
ZIP
ZIP
ZIP

```
 1 REM prueba
10 LET p=PEEK 23635+256
   *PEEK 23636
15 LET v=PEEK 23627+256
   *PEEK 23628
20 LPRINT "p=";p,"v=";v:LPRINT
25 FOR k=p TO v
30 LET x=PEEK k
35 LPRINT x;" ";:IF x=13 THEN LPRINT
   :LPRINT
40 NEXT k
45 STOP
```

Veamos ahora el significado de las distintas instrucciones:



línea 10: calcula el puntero p de principio de programa;

línea 15: calcula el puntero v de principio de variables;

línea 20: imprime el valor de p y v;

líneas 25-40: es un bucle que imprime el contenido de los bytes desde p a v, volviendo al principio cuando un byte contiene 13 (que es el código de ENTER);

línea 45: fin del programa.

Ejecutándolo, obtendrás como salida algo parecido a lo siguiente:

```
p=23755          v=23935

0 1 7 0 234 112 114 111 118 97 13

0 10 39 0 241 112 61 190 50 51 54
50 55 14 0 0 75 92 0 43 50 53
54 14 0 0 0 1 0 42 190 50 51 54
50 56 14 0 0 76 92 0 13
```

....

y así sucesivamente.



Intentemos comprender ahora qué significan estas series de números.
El programa ocupa 23935–23755=180 bytes: esta es la memoria total empleada

en las instrucciones. Cada grupo de valores es la representación numérica con la que tu Spectrum ha memorizado las instrucciones. Por ejemplo, en correspondencia a 1 REM prueba, encontraremos:
0 1, que es el número de línea: 0*256+1=1

7 0, 0*256+7=7, que es la longitud del texto de la instrucción + ENTER

234 es el código ASCII de REM

112 114 111 118 97, son los códigos ASCII de los caracteres de la palabra «prueba», escritos en minúsculas

13, código de ENTER.

En cambio, por lo que respecta a la línea

```
10 LET p=PEEK 23635+256*PEEK 23636
```

tendremos:
0 10, número de línea 0*256+10=10

39 0, 0*256+39=39 es la longitud del texto + ENTER

241, es el código ASCII de LET

112, es el código ASCII de p minúscula

61, es el código ASCII de =

190, es el código ASCII de PEEK

50 51 54 51 53, son los códigos ASCII de las cifras del número 23635 (que se emplean cuando se lista)

14, marcador de principio de número en formato exponencial (empleado por el ordenador durante la ejecución)

0 0 83 92 0, son los cinco bytes de la representación del número entero 23635

43, es el código ASCII de +

50 53 54, códigos ASCII de las cifras del número 256

14, marcador de principio de número en formato exponencial

0 0 0 1 0, son los cinco bytes de la representación del número entero 256

42, código ASCII de *

190, es el código ASCII de PEEK

50 51 54 51 54, son los códigos ASCII de las cifras del número 23636

14, marcador de principio de número en formato exponencial

0 0 84 92 0, son los cinco bytes de la representación exponencial del número

entero 23636.

Las demás líneas del programa pueden analizarse de la forma que acabamos de ver: te lo dejamos como ejercicio. A cambio te proponemos otro interesante programa, cuya tarea es la de imprimir las palabras reservadas que el BASIC convierte en TOKEN, es decir, reducidas a un solo código numérico.
En la memoria ROM del Spectrum, desde el byte con la dirección 150 hasta el byte con la dirección 516, están contenidas las descripciones de los diversos TOKEN empleados por el programa.

```
10 REM token
12 LPRINT "Dec. hexadec."
14 LPRINT "ASCII ASCII TOKEN":LPRINT
16 LET n=165:LET sw=0
18 LET h$="0123456789ABCDEF"
20 FOR a=150 TO 516
22 LET b=PEEK a
23 IF sw <> 0 THEN GO TO 40
24 LET bh=INT(n/16):LET bl=n-bh*16
26 LET b$=h$(bh+1)+h$(bl+1)
28 LET C$=STR$ n+" ":LET C$=C$(1 TO 4)
30 LPRINT C$;" ";b$;" ";:LET SW=1
40 IF b<128 THEN PRINT CHR$ b;:GO TO 60
55 LPRINT CHR$(b-128):LET n=n+1:LET
SW=0
60 NEXT A
```

Ejecutando el programa aparecerán en tu impresora (o en pantalla, sustituyendo los LPRINT por PRINT) las correspondencias entre los números y las palabras reservadas.

Las interrupciones

La CPU usa el área del stack para numerosas operaciones, entre ellas el manejo de las interrupciones.
La comunicación entre la CPU y los periféricos puede realizarse mediante dos distintas técnicas. La primera

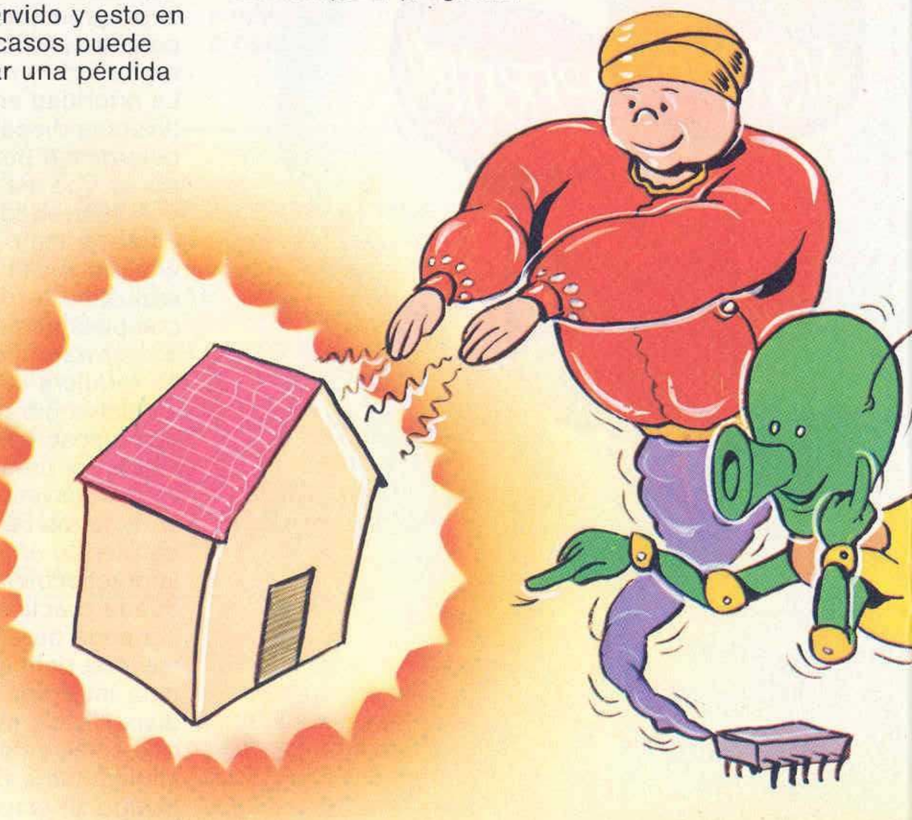




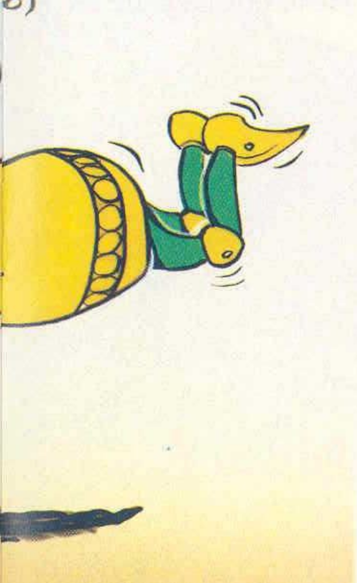
toma el nombre de «polling» (interrogación cíclica): la CPU interroga cíclicamente, según un orden preestablecido, a los dispositivos externos, empleando un programa que lee el estado de los mismos. Cuando uno de los dispositivos esté listo para transmitir o recibir un dato, la CPU manda el programa necesario para la operación requerida.
La prioridad entre los diversos dispositivos se determinan por el orden por el que estos últimos son interrogados.
Esta técnica presenta una única ventaja: la de realizarse completamente a través de software, por lo que no requiere circuiteria adicional para las comunicaciones. Sin embargo, presenta varias desventajas, que en algunos casos la convierten en impracticable: la CPU queda prácticamente ocupada durante la mayoría del tiempo sólo para interrogar dispositivos, mientras que solamente una mínima parte de este tiempo se emplea para la comunicación propiamente dicha.

Además el tiempo de respuesta de la CPU a las peticiones de los dispositivos puede ser en algunos casos demasiado largo: si por ejemplo hay varios dispositivos y uno de ellos requiere una operación de E/S inmediatamente después de haber sido interrogado, tendrá que esperar bastante antes de ser servido y esto en algunos casos puede comportar una pérdida de datos.

La segunda técnica, que elimina estas desventajas, es la de las interrupciones.
Aquí son los mismos dispositivos los que señalan a la CPU la petición de una operación de E/S, enviando una señal, llamada precisamente interrupción, que le pide a la CPU que interrumpa el programa que esté ejecutando, para efectuar la operación de E/S. El servicio de interrupciones se efectúa mediante un salto a la rutina que toma el nombre de rutina de servicio de las interrupciones, y esta es la que se ocupa de



ejecutar la operación requerida.
Esta técnica presenta muchas analogías con la ejecución de subrutinas, con la diferencia de que la ejecución de las rutinas de servicio ocurre en momentos no previsibles por el programa, y que dependen de las exigencias de los dispositivos exteriores.
Es evidente que la técnica de las interrupciones elimina las desventajas antes indicadas. El tiempo de respuesta sólo está limitado por la rapidez de la CPU para transferir el control de una a otra zona de la memoria; además, la unidad central puede así dedicarse a otros programas, empleando su tiempo de una forma más eficaz.
La diferencia entre «polling» e interrupciones es la misma que habría entre abrir la puerta de casa a intervalos de tiempo prefijados para ver si hay alguien y la de contestar, en cambio, cuando suena el timbre.
La pérdida de tiempo en el primer caso es evidente, así como la incomodidad del servicio para aquél que, deseando entrar, tenga que esperar al siguiente control para poder hacerlo. Sin embargo, tenemos que decir que para la gestión de las interrupciones, será necesario, además del dispositivo (el timbre) que indique la petición de comunicación, diponer de un hardware mucho más complicado.
En primer lugar se necesitan una o más líneas de la CPU dedicadas a la recepción de las señales de interrupción, y además es necesario que la misma interrupción se haga reconocer por la CPU, y esto hace también más complejas las conexiones; finalmente, aunque no en todos los casos, es necesaria una circuitería que permita el servicio a distintos dispositivos, en base a prioridades prefijadas.
Dado que la técnica de las interrupciones es tan ventajosa desde un punto de vista práctico, casi todos los fabricantes de ordenadores, excluyendo casos muy especiales, recurren a ella habitualmente.
Además, la posibilidad de empleo de las interrupciones se extiende hasta los programadores normales gracias a instrucciones especiales de las que está dotada la CPU. Por lo tanto, es interesante conocer la importancia de las interrupciones por dos razones:
1) cualquiera puede usarlas en sus programas en Assembler;
2) su funcionamiento ayuda a comprender todas aquellas acciones que realiza el ordenador sin que haya una intervección directa del programador.



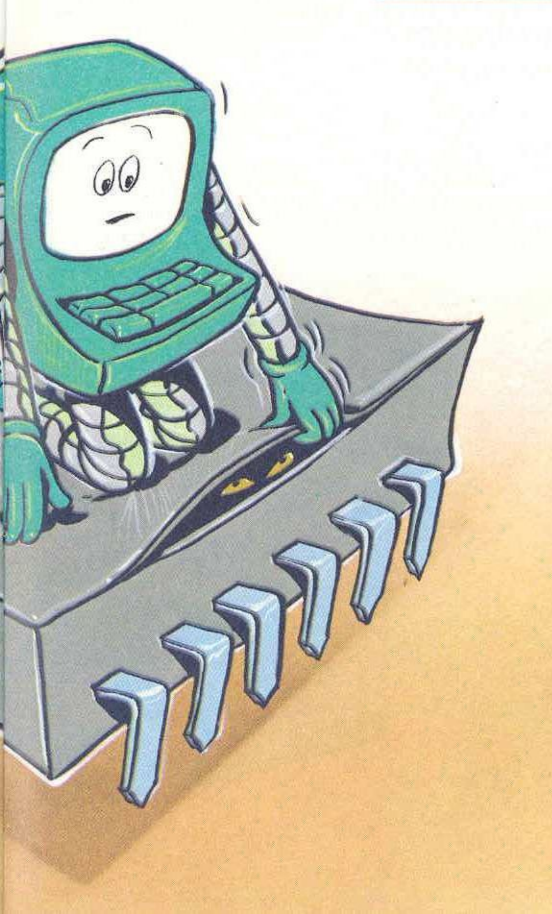
Usar la ROM

Existen muchas rutinas del sistema que permiten a un programador experto ahorrar el tiempo y el trabajo de volverlas a escribir.
Los diseños de ordenadores estructuran las rutinas de forma tal que pueden ser consideradas como subprogramas normales, requeribles en cualquier momento tanto desde BASIC como desde el Assembler. La ventaja de esta solución es más que evidente: se evita a los programadores tener que enfrentar cada vez los mismos problemas, por ejemplo, la visualización de resultados, permitiéndoles concentrarse sobre un problema específico en lugar de sobre los problemas generales.
Además, las rutinas del sistema, incluidas en la ROM, y por lo tanto, imborrables, han sido escritas y comprobadas por programadores profesionales, lo que garantiza su seguro funcionamiento.
Para poder usar una cualquiera de estas rutinas, lo único que hay que conocer es su dirección inicial, además de, naturalmente, los registros del microprocesador que sean afectados por esta rutina. Existen muchos manuales que describen con notable exactitud todas estas rutinas; nosotros trataremos de las principales, explicando como usarlas en los programas que es su aspecto más importante.
Cada una de estas rutinas posee su propio nombre mnemónico, asignado normalmente por la casa fabricante, lo que permite distinguirla inmediatamente de las demás. Así pues, la siguiente tabla contiene, además de la dirección



inicial, el nombre de cada rutina.

NOMBRE	DIRECCION	OBJETIVO
PRINTOUT	09F4	visualiza en pantalla
START	0000	inicializa el sistema
KEYBOARD	02BF	comprueba el teclado
BEEPER	03B5	hace sonar el altavoz
SAVE	0605	rutina de SAVE
LOAD	0808	rutina de LOAD
CLS	0D6B	limpia la pantalla
NEW	11B7	rutina de NEW
PLOT	22DC	dibuja un punto
SCROLL	0DFE	scroll de una línea



Como ejemplo de uso de una de estas rutinas, veremos de qué forma es posible imprimir algo en pantalla.
De la tabla anterior se deduce que la rutina dedicada a la visualización de caracteres en pantalla es PRINTOUT; únicamente necesita que, antes de llamarla, el código ASCII del carácter a visualizar esté situado en el acumulador.

Por lo tanto, tendremos que memorizar en el acumulador un código determinado y llamar a la rutina tantas veces como el número de caracteres que deseamos imprimir. Para imprimir la palabra «chao», podremos entonces escribir:

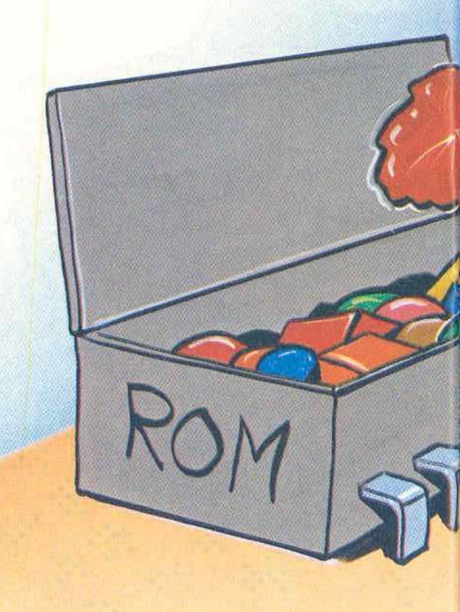
```
LD A,43H        ;ASCII de "C"
CALL 09F4

LD A,48H        ;ASCII de "H"
CALL 09F4

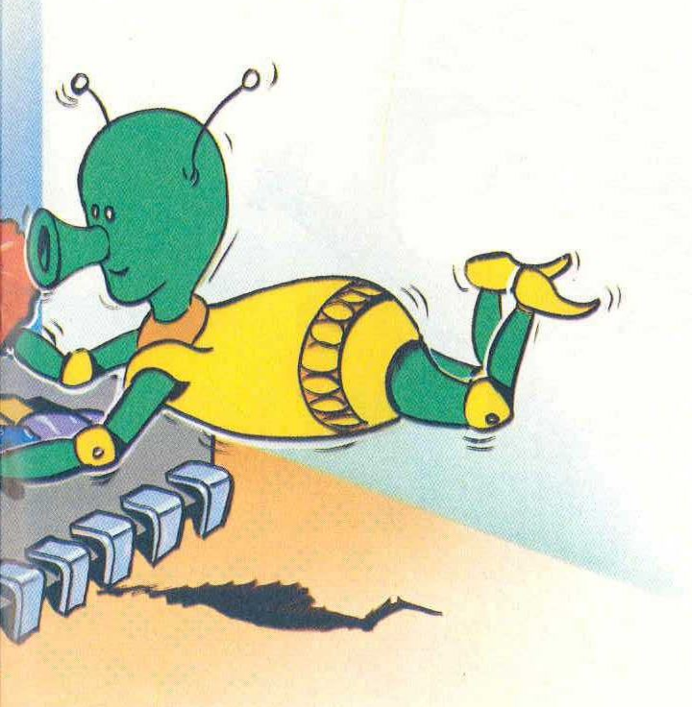
LD A,41H        ;ASCII de "A"
CALL 09F4

LD A,4FH        ;ASCII de "O"
CALL 09F4
LD DE,0105H
LD HL,066AH
CALL 03B5
RET
```

A la operación de impresión le hemos añadido, al final, una musiquilla de altavoz, recurriendo a la subrutina BEEPER. Puesto que esta rutina requiere que en los registros DE y HL estén los valores numéricos correspondientes a la nota a tocar, antes de CALL hemos introducido dichos valores (y habrás podido ver también que no es posible usar una



rutina en ROM conociendo tan sólo su dirección).
Así, el hecho de conocer la existencia de la rutina PRINTOUT nos ha evitado cualquier preocupación por lo que respecta a la salida de los resultados en pantalla. La llamada se realiza sencillamente a través de la habitual instrucción CALL, y es una ulterior demostración de que en la ROM las rutinas están escritas en forma de subprogramas.



La velocidad de ejecución

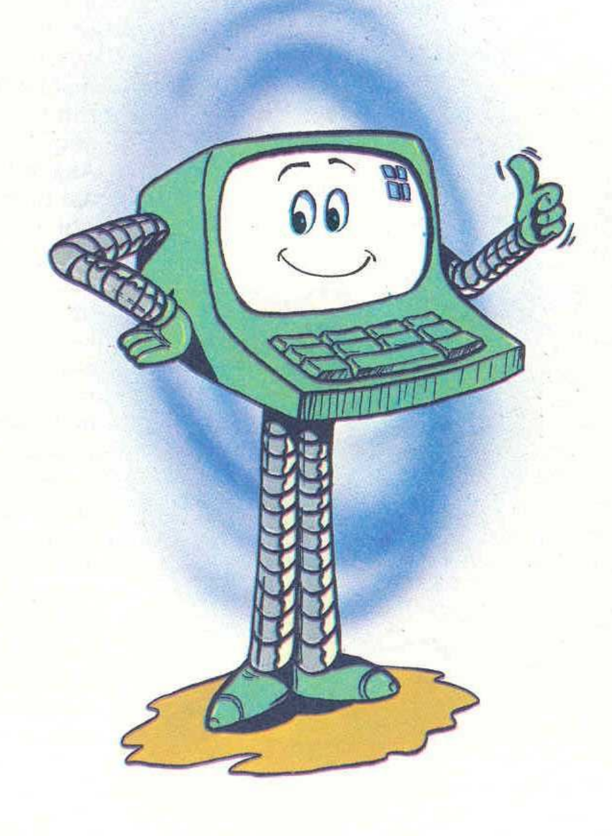
Ahora nos proponemos resolver el siguiente problema: colocar en orden creciente unos valores dispuestos al azar pertenecientes a una determinada matriz numérica.
Por lo tanto, deseamos escribir un programa de ordenación. Ya hemos tocado este problema en una de nuestras lecciones; hoy sin embargo, propondremos dos soluciones distintas pero idénticas: la primera en BASIC y la segunda en Assembler.Y lo haremos así para que puedas ver por tí mismo la notable diferencia de velocidad existente entre uno y otro lenguaje.
La técnica que emplearemos para realizar el trabajo es muy sencilla, y se puede resumir así:
– Sobre una matriz de N números realizaremos comparaciones cíclicas, en total N–1 veces.
– Durante el primer ciclo compararemos el primer elemento con todos los demás; cuando encontremos un número menor que él

continuaremos la comparación con éste. Al final del bucle llevamos a la primera posición el número más pequeño que hemos encontrado, intercambiándolo con aquel que allí había salvo que éste fuera menor. Esta operación se programa fácilmente sirviéndose de los indices de la matriz.

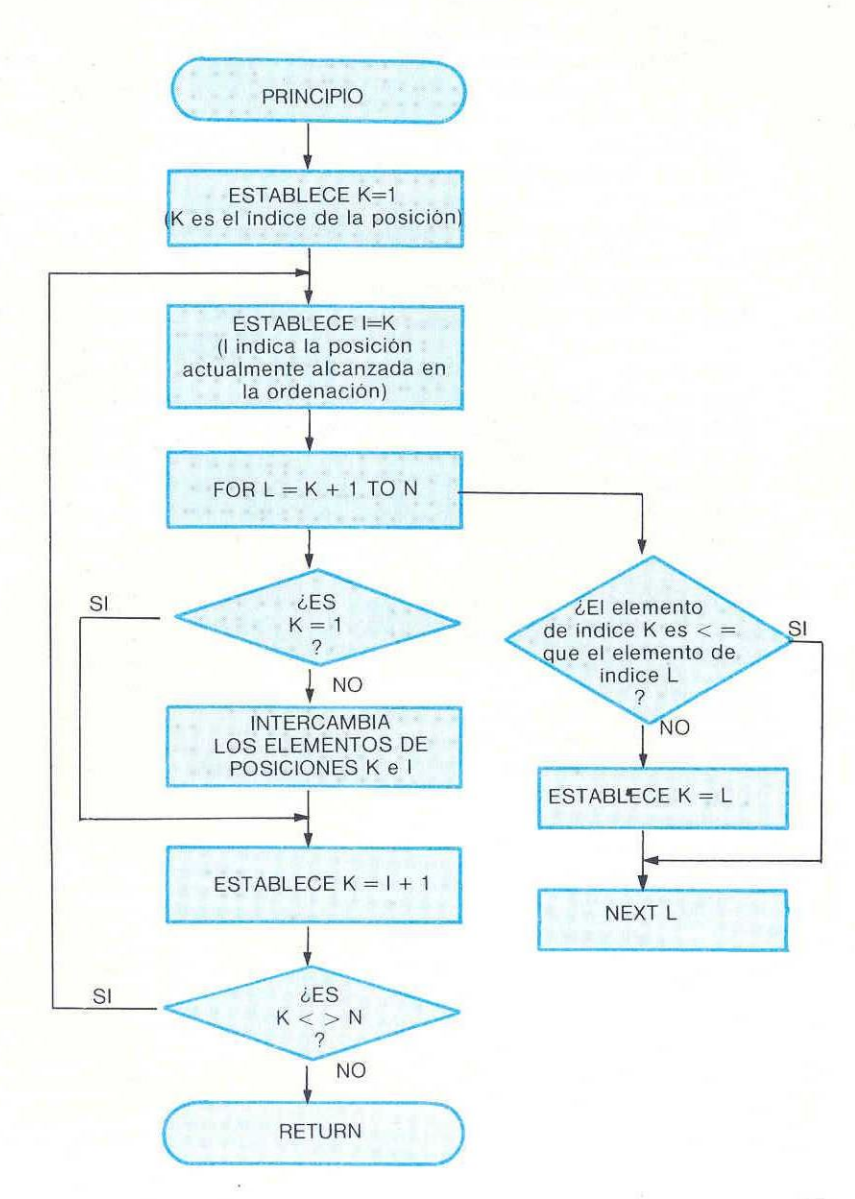
– Cada bucle coloca en su sitio un elemento a partir del indice menor; en consecuencia cada bucle disminuye en 1 el número de los elementos a comparar.

– Al final, después de haber ejecutado N–1 bucles, todos los elementos estarán ordenados.

Dado que el corazón del programa está constituido por la fase de ordenación, utilizaremos una subrutina para desarrollar esta tarea (haciendo esto podremos utilizar el mismo programa principal, tanto para la versión en BASIC como para aquélla en Assembler).



El diagrama de flujo de la rutina de ordenación es el siguiente:



El programa completo es éste:

```
  10 REM petición de datos iniciales
  15 CLS: INPUT "¿Cuántos números?: "; N
  20 IF N > 255 THEN GO TO 15
  30 REM dimensiona la matriz y saca los
     números al azar
  40 DIM V(N)
  50 FOR K=1 TO N
  60 LET V(K)=INT(RND(0)*5000)
  70 NEXT K
  80 GO SUB 500:REM visualiza la matriz
     "desordenada"
  90 GO SUB 1000:REM ordena la matriz
 100 GO SUB 500:REM visualiza la matriz
     "ordenada"
 110 STOP
 500 FOR K=1 TO N
 510 PRINT (K)
 520 NEXT K
 530 PRINT:PRINT
1000 REM rutina de ordenación
1010 LET K=1
1015 LET I=K
1020 FOR L=K+1 TO N
1030 IF V(K)<=V(L) THEN GO TO 1050
1040 LET K=L
1050 NEXT L
1060 IF K=1 THEN GO TO 1080
1070 LET C=V(I):LET V(I)=V(K):LET V(K)=C
1080 LET K=I+1:IF K<>N THEN GO TO 1015
1090 RETURN
```

Veamos brevemente el funcionamiento del programa:
líneas 10-20: se pide el número de elementos a ordenar; si este número supera el 255 (hemos puesto como límite este valor), la petición se repite.
Líneas 30-40: se dimensiona la matriz.
Líneas 50-70: se sacan al azar los N números a insertar en la matriz.
Líneas 80-100: primero, se visualiza la matriz, después se ordena y, finalmente, se visualiza de nuevo para comprobar la ordenación.
Líneas 500-530: rutinas de visualización de la matriz.
Ejecutando el programa y estableciendo un número de elementos igual a 255, obtendrás la ordenación de la matriz en un tiempo medio (que depende de la secuencia de números sacados al azar) que se aproxima a los ochocientos segundos (cerca de 13 minutos).
Tiempos bien distintos obtendremos ejecutando el programa con la misma rutina de ordenación escrita en Assembler. He aquí las líneas de programa que

tendrás que añadir a las ya vistas para poder elegir entre ordenación BASIC y Assembler.

```
   5 CLEAR 59999
  25 INPUT "¿Ordenación BASIC o Assembler (B/A)?";R$
  85 IF R$="A" OR R$= "a" THEN GO SUB 2000: GO TO 100
2000 RESTORE 2090
2010 FOR K=60000 TO 60102:READ L:POKE K,L:NEXT K
2020 REM prepara la llamada a la rutina
2030 LET C=PEEK 23627+256*PEEK 23628
2040 LET Y1=PEEK(C+3)-1:POKE 65533, Y1:POKE 65532, Y1
2050 LET Y1=C+41:LET Y2=INT(Y1/256):LET Y3=Y1-Y2*256
2060 POKE 65528,Y3:POKE 65529,Y2
2070 RANDOMIZE USR 60000
2080 RETURN
2090 DATA 42,248,255,229,209,205,193,234,19
2100 DATA 26,35,150,40,6,56,16,229,209,24,12
2110 DATA 43,27,26,150,40,8,56,6,229,209,24,2
2120 DATA 43,27,205,193,234,229,33,252,255,53
2130 DATA 225,40,2,24,217,42,248,255,124,146
2140 DATA 32,6,125,147,32,2,24,16,78,35,70,43
2150 DATA 26,119,35,19,26,119,27,121,1,19,120
2160 DATA 18,42,248,255,205,193,234,34,248,255
2170 DATA 33,253,255,53,32,1,201,126,43,119
2180 DATA 24,159,35,35,35,35,35,201
```

La línea 5 baja el TOP de la memoria BASIC, para poder cargar la rutina en código máquina. Las líneas 25 y 85 sirven para poder elegir la ordenación que se desea (BASIC o Assembler).
Las líneas 2000-2180 introducen en la memoria los códigos de la rutina, inicializando y preparando algunas localizaciones de memoria necesarias

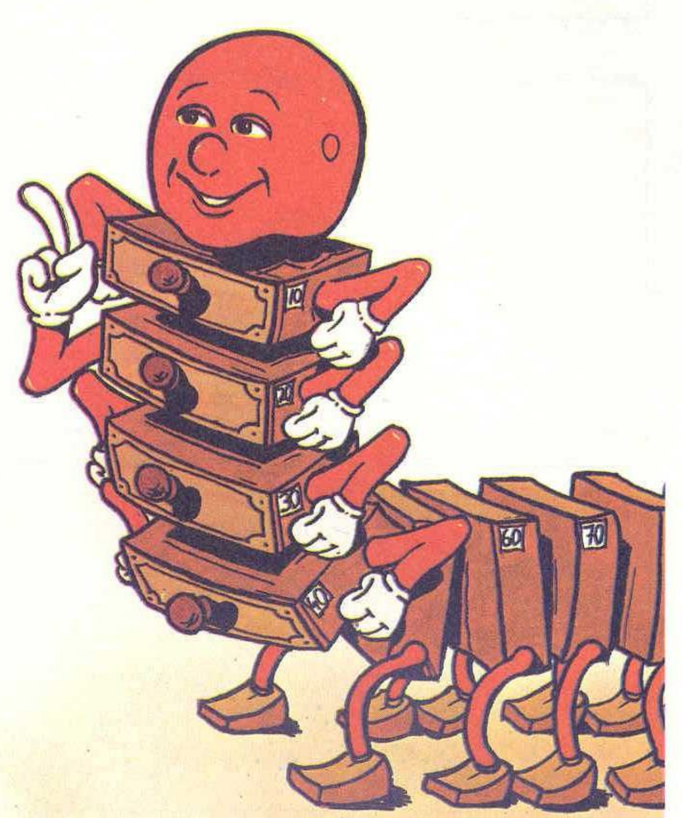
para el funcionamiento del programa Assembler.
En especial, comentaremos que la línea 2070 es la que pone en marcha la ejecución de la rutina.
El tiempo medio de ordenación obtenible trabajando con el programa en código máquina es de aproximadamente 2 segundos: en este caso (aunque no siempre ocurra esto) la ventaja del Assembler respecto al BASIC es de... ¡1 a 400!

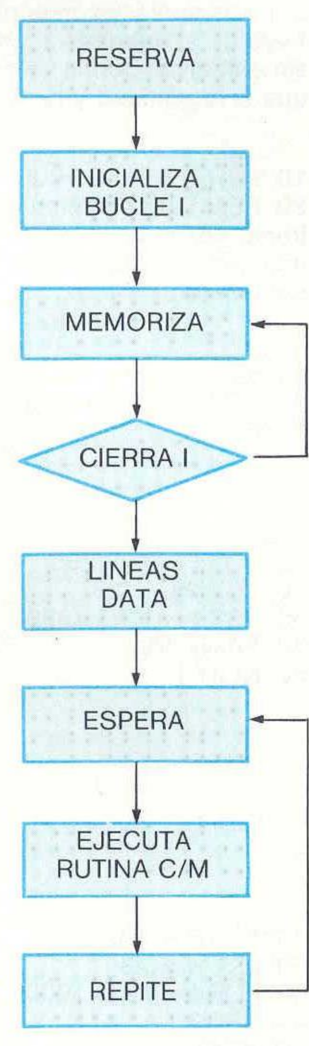
Print sin PRINT

Este es un buen ejercicio y una excelente demostración de cómo emplear el sistema operativo de la ROM saltándose el intérprete BASIC.
He aquí la parte C/M.

LD A,2	Prepara el registro A para...
CALL 1601	...activar el canal del display
LD A,(23560)	Introduce LASTK (última tecla)
RST 10	Llama a la rutina de impresión
RET	Vuelve

```
10  CLEAR 29999
20  FOR I = 30000 TO 30009
30  READ X
40  POKE I,X
50  NEXT I
60  DATA 62,2,206,1,22,58,8,92,215,201
70  PAUSE 0
80  RANDOMIZE USR 30000
90  GO TO 70 LM: 30000 X : VAR. AYUDA
```





I: VAR. DE CONTROL

EJERCICIOS

¿Cuáles serán los más rápidos de los siguientes programas?
Trata de averiguarlo cronómetro en mano, y de explicarte la razón. Recuerda, sin embargo, que la velocidad puede ser importante y a veces esencial, pero que la legibilidad y la claridad del listado lo son todavía más.

```
10 FOR P = 1 TO 1000
20 REM Este comentario
frena; por lo tanto, salvo
que sea indispensable,
será mejor omitirlo

30 NEXT P
10 FOR P = 1 TO 1000:
NEXT P
```

```
10 POKE 23692,255
20 LET N$ = "23"
30 FOR I = 1 TO 1000
40 PRINT N$
50 NEXT !
```

```
10 POKE 23692,255
20 LET N = 23
30 FOR I = 1 TO 1000
40 PRINT N;
50 NEXT I
```